Avoiding Numerical Cancellation in the Interior Point Method for Solving Semidefinite Programs

Sturm, J.F.

Publication date:
2001

No. 2001-27

AVOIDING NUMERICAL CANCELLATION IN THE INTERIOR POINT METHOD FOR SOLVING SEMIDEFINITE PROGRAMS

By Jos F. Sturm

April 2001

ISSN 0924-7815

Discussion paper

Avoiding Numerical Cancellation in the Interior Point Method for Solving Semidefinite Programs

Jos F. Sturm
Department of Econometrics, Tilburg University, The Netherlands.
j.f.sturm@kub.nl
http://fewcal.kub.nl/sturm

March 2, 2001

Abstract

The matrix variables in a primal-dual pair of semidefinite programs are getting increasingly ill-conditioned as they approach a complementary solution. Multiplying the primal matrix variable with a vector from the eigenspace of the non-basic part will therefore result in heavy numerical cancellation. This effect is amplified by the scaling operation in interior point methods. In order to avoid numerical problems in interior point methods, we therefore propose to maintain the matrix variables in a product form. We discuss how the factors of this product form can be updated after a main iteration of the interior point method with Nesterov-Todd scaling.

Keywords: Semidefinite Programming.

AMS subject classification: 90C22, 90C20.

JEL codes: C61, C63.

1 Introduction

This paper addresses numerical issues in interior point methods for semidefinite programming, particularly focusing on the final iterates. The numerical issues that we encounter belong to the category of *numerical cancellation*. In general, the phenomenon of numerical cancellation refers to the following situation: the computer adds two numbers, say α and β, and the result is much smaller in magnitude, i.e. $|\alpha + \beta|/(|\alpha| + |\beta|) << 1$. For instance, in a floating point system with 4 digits of accuracy, the computation '`-0.1350 + 0.1357 = 0.7E-3`' suffers from numerical cancellation, since the result has merely one signifant digit: we do not know whether it is say `0.65E-3` or `0.74E-3`, because '`-0.1350`' and '`0.1357`' are merely 4-digit floating point representations of real values.

In linear programming, we know that some of the nonnegative decision variables will have to approach zero in order to obtain optimality; these are the so-called non-basic variables. In

semidefinite programming, the eigenvalues of a symmetric matrix variable X are restricted to be nonnegative, and some of these nonnegative eigenvalues approach zero for near optimal solutions. In fact, associated with a semidefinite programming problem is a matrix Q_N of full column rank such that $X^{(k)}Q_N \to 0$ for any solution sequence $X^{(1)}, X^{(2)}, \ldots$ that approaches optimality [7, 11]. However, the entries in the matrix $X^{(k)}$ itself will in general *not* approach zero. Therefore, the computation '$X^{(k)}Q_N$' will increasingly suffer from numerical cancellation when optimality is approached. This is also true for the computation 'XZ', where Z is a nearly optimal dual slack variable associated with a nearly optimal matrix variable X. Particularly dangerous are computations involving X^{-1}, since X is getting nearly singular, and the small eigenvalues cannot be reliably computed from X. However, these type of computations typically appear in interior point methods, as part of the so-called scaling operation. The loss of accuracy due to numerical cancellation will be even more pronounced if the sequence $X^{(1)}, X^{(2)}, \ldots$ is unbounded (i.e. the entries become arbitrarily large), as is true for problems in which the optimal value cannot be attained.

We remark that the problems of numerical cancellation as discussed so far do not occur in linear programming, because there the quantities that approach zero are stored as separate entries, viz. the non-basic variables. Therefore, we propose to store the primal and dual iterates (X and Z) in a *product form* that allows us to maintain the contributions from all eigenvalues with high accuracy. The technique has been implemented as Version 1.04 of the popular semidefinite programming solver SeDuMi [18].

Numerical issues in semidefinite programming have been addressed in several research papers [1, 10, 23]. Todd, Toh and Tütüncü [23] investigated the so-called AHO, HRVW/KSH/M, and NT search directions in a numerical experiment, using their SDPT3 software [25]. By and large, they obtained the highest accuracy with the AHO direction. Alizadeh, Haeberly and Overton [1] provided a possible explanation why the AHO direction may have better numerical performance in the final stage of the interior point method than the HRVW/KSH/M and NT search directions. Very recently, Kruk [10] observed that the Gauss-Newton method may suffer less from numerical problems than the interior point method, and he also provides a possible explanation. But, unlike the interior point approach, the Gauss-Newton method is not known to have polynomial convergence, and the computational experience on practical semidefinite programming models is still relatively limited.

The organization of the paper is as follows. In Section 2, we discuss the definitions and notational conventions which are used in the remaining sections. We also state some well known facts from the interior point method for semidefinite programming, especially concerning the predictor direction with Nesterov-Todd scaling. In Section 3 we propose to maintain the iterates of the interior point method in the so-called V-space product form of Sturm and Zhang [20] with the special requirement that the U_d-factor is the Cholesky factor of the Nesterov-Todd scaling point [15, 16]. In Section 4, we provide a compuational method to update the V-space factors between two successive main iterations of the interior point method. In Section 5, we discuss how the V-space factors can be used to compute in a more accurate fashion the $AP(d)A^{\mathrm{T}}$-matrix that appears in the scaled normal equations system which defines the interior point search directions. The actual accuracy that can be obtained by the interior point solver depends on the weakest chain in the link: the improved accuracy that we obtain in one part of the computations can be lost elsewhere in the process.

Therefore, we identify miscellaneous sources of numerical problems in Section 6, and we provide possible remedies. Numerical results are reported in Section 7. The paper in concluded in Section 8.

2 Preliminaries

We consider semidefinite programming problems in the following standard form:

$$\begin{array}{ll} \text{minimize} & \text{tr } CX \\ \text{such that} & \text{tr } A_i X = b_i \text{ for } i = 1, 2, \ldots, m \\ & X \text{ is symmetric and positive semidefinite,} \end{array} \tag{1}$$

where $A_1, A_2, \ldots, A_m$ and C are given $\nu \times \nu$ matrices, b is a vector with components $b_1, b_2, \ldots, b_m$, and the decision variable is an $\nu \times \nu$ matrix X.

The standard $\text{vec}\,(\cdot)$ operator [9] stacks the columns of a matrix into a long vector, i.e.

$$\text{vec}\,(X) = \text{vec}\,(\begin{bmatrix} x_1 & \cdots & x_\nu \end{bmatrix}) = \begin{bmatrix} x_1 \\ \vdots \\ x_\nu \end{bmatrix},$$

where $x_i \in \Re^\nu$, $i = 1, \ldots, \nu$, are the columns of X.

Letting $a_i = \text{vec}\,(A_i)$, $i = 1, \ldots, m$, $A := \begin{bmatrix} a_1, & \cdots, & a_m \end{bmatrix}^{\mathrm{T}}$, and $c := \text{vec}\,(C)$, we can reformulate the standard semidefinite programming problem as a conic linear program in $\Re^n$ with $n = \nu^2$ as follows:

$$\inf\{c^{\mathrm{T}} x \mid Ax = b,\ x \in \text{PSD}(\nu)\}. \tag{2}$$

The decision variable is now $x = \text{vec}\,(X) \in \Re^n$. The convex cone $\text{PSD}(\nu)$ is defined as follows:

$$\text{PSD}(\nu) := \{\, \text{vec}\,(X) \mid X \in \Re^{\nu \times \nu} \text{ is symmetric and positive semidefinite}\}.$$

The dual cone of $\text{PSD}(\nu)$ in $\Re^n$ is then

$$\text{PSD}(\nu)^* = \{\, \text{vec}\,(Z) \mid Z \in \Re^{\nu \times \nu},\ Z + Z^{\mathrm{T}} \text{ is positive semidefinite}\}.$$

Associated with (2) is a dual problem, viz.

$$\sup\{b^{\mathrm{T}} y \mid c - A^{\mathrm{T}} y \in \text{PSD}(\nu)^*\}. \tag{3}$$

The vector of dual decision variables is $y \in \Re^m$.

We may replace 'C' by $(C + C^{\mathrm{T}})/2$ and similarly 'A_i' by $(A_i + A_i^{\mathrm{T}})/2$, $i = 1, 2, \ldots, m$, without affecting problems (2) and (3). Therefore, we will assume without loss of generality that the matrices A_i, $i = 1, \ldots, m$ and C are symmetric. Under this assumption, we have

$$c - A^{\mathrm{T}} y \in \text{PSD}(\nu)^* \iff c - A^{\mathrm{T}} y \in \text{PSD}(\nu).$$

Given a dual solution y we define the dual slack as $Z = C - \sum_{i=1}^{m} y_i A_i$, or $z = \text{vec}\,(Z) = c - A^{\mathrm{T}} y$ in vectorized form.

Given matrices that are identified by upper-case symbols X, Z and C, we implicitly define the lower case symbols x, z and c as $x := \text{vec}\,(X)$, $z := \text{vec}\,(Z)$, and $c := \text{vec}\,(C)$, We will use the matrix form (1) and vector form (2) of a semidefinite program interchangeably, depending on the context.

2.1 Path-Following Direction

In each iteration of the (feasible) primal-dual interior point method, a primal feasible solution X and a dual feasible solution Z are computed, such that X and Z are positive definite. Whenever such solutions exist, it holds that positive *semi*-definite feasible solutions X and Z are optimal if and only if $XZ = 0$. We refer to Vandenberghe and Boyd [26] and Todd [21, 22] for general introductions to semidefinite programming. The distance to optimality for a feasible solution is measured by the duality gap, which is $c^{\mathrm{T}} x - b^{\mathrm{T}} y = x^{\mathrm{T}} z = \text{tr}\, XZ$.

We let $\lambda_{\min}(XZ)$ denote the smallest eigenvalue of XZ, which is real since $XZ \sim X^{1/2} Z X^{1/2}$. If (X, y, Z) is a given iterate in the interior point method, then X and Z are positive definite, and $\lambda_{\min}(XZ)/(x^{\mathrm{T}} z)$ is bounded from below by a positive constant which is independent of the duality gap; this is the so-called centrality condition. In fact, on the central path we have $XZ = (x^{\mathrm{T}} z/\nu) I$, where I denotes the identity matrix. The centrality property is important for our numerical investigation.

A main iteration of the interior point method consists of computing a search direction that is added to the current iterate with a certain step length $t > 0$, yielding the next iterate:

$$(X^{\text{new}}, y^{\text{new}}, Z^{\text{new}}) = (X, y, Z) + t(\Delta X, \Delta y, \Delta Z). \tag{4}$$

The search direction $(\Delta x, \Delta y, \Delta z)$ is implicitly defined by a system of equations, as follows:

$$\begin{cases} \Delta x + P(d)\Delta z = r \\ A\Delta x = 0 \\ A^{\mathrm{T}}\Delta y + \Delta z = 0, \end{cases} \tag{5}$$

The system depends on an invertible $n \times n$ matrix '$P(d)$' and a vector $r \in \Re^n$, which depend not only on the iterate, but also on the specific algorithmic choices of the interior point method. E.g. setting $r = -x$ corresponds to the so-called *predictor* (or: primal-dual *affine scaling*) direction. Various approaches in semidefinite programming have led to different possible choices for '$P(d)$', see the surveys by Todd [21, 22], where this matrix is denoted as '$\mathcal{E}^{-1}\mathcal{F}$'. On the central path, we have in most approaches that $P(d)$ is a multiple of $X \otimes X$ or, equivalently, $Z^{-1} \otimes Z^{-1}$. Here, '$\otimes$' denotes the standard Kronecker product [9], i.e.

$$(D \otimes D)\,\text{vec}\,(X) = \text{vec}\,(DXD^{\mathrm{T}}).$$

We remark from (5) that

$$0 = AP(d)(A^{\mathrm{T}}\Delta y + \Delta z) = AP(d)A^{\mathrm{T}}\Delta y + A(r - \Delta x) = AP(d)A^{\mathrm{T}}\Delta y + Ar,$$

yielding

$$AP(d)A^{\mathrm{T}}\,\Delta y = -Ar. \tag{6}$$

Notice that for the predictor direction with $r = -x$ we have $-Ar = b$, and (6) can be further simplified to

$$AP(d)A^{\mathrm{T}}\,\Delta y = b. \tag{7}$$

Once the Δy direction is known, the Δz and Δx directions then follow easily as

1. $\Delta z = -A^{\mathrm{T}}\Delta y$, and
2. $\Delta x = r - P(d)\Delta z$.

The main computational effort in the interior point method lies therefore in solving a system of the form (7). From a numerical point of view, evaluating '$r - P(d)\Delta z$' is also a challenge (even in the case of linear programming). Namely, if the iterates converge then Δx $(= r - P(d)\Delta z)$ approaches zero, whereas r does not; hence the accuracy in Δx suffers from numerical cancellation. However, the other sources of numerical problems that we discuss in this paper are much more severe.

In actual implementations of the interior point method, the right hand side in (7) may not be exactly the b vector, because

- an infeasible interior point method is used, i.e. Ax is not necessarily equal to b, see Lustig, Marsten and Shanno [12], or
- a self-dual embedding is used, see Ye, Todd and Mizuno [27], and
- the direction is not simply a predictor direction, but includes e.g. a second order correction, see Mehrotra [13].

Nevertheless, for the purpose of this paper it suffices to focus on the system (7). This systems stays the same throughout the interior point process, except for the parameter d. Thus, if d is not sufficiently accurate then solving (7) becomes pointless due to the 'garbage in, garbage out' effect. We will now finally address the following issue: *What are $P(\cdot)$ and d ?*

2.2 Nesterov–Todd Scaling

We use the notation from Euclidean Jordan algebra [5] to let the $n \times n$ matrix $P(d)$ denote the *quadratic representation* of $d \in \Re^n$. This means for $d = \mathrm{vec}\,(D)$ with $D = D^{\mathrm{T}}$ that

$$P(d) = P(\,\mathrm{vec}\,(D)) = D \otimes D.$$

By construction, $P(d)\mathrm{PSD}(\nu) = \mathrm{PSD}(\nu)$ for any $d \in \Re^n$. Furthermore, if D is positive definite then $P(\,\mathrm{vec}\,(D))$ is positive definite and $P(\,\mathrm{vec}\,(D))^{-1} = P(\,\mathrm{vec}\,(D^{-1}))$, see [9].

For the vector $d = \mathrm{vec}\,(D)$, $D = D^{\mathrm{T}}$, we will use the *Nesterov-Todd scaling* [15, 16], also known as *NT-scaling*, which is defined as the unique positive definite solution to

$$X = DZD, \tag{8}$$

for given positive definite iterates X and Z. In fact, D is the metric geometric mean [2, 23] of X and Z^{-1}. Vectorizing (8) yields the equivalent characterization $x = P(d)z$.

3 V-Space Product Form

In the final stage of the interior point method, accurate information on the small(est) eigenvalues of X and Z can usually not be obtained from the floating point representations of X and Z as these matrices become increasingly ill conditioned. Therefore, we propose to maintain these iterates in a specific product form, which we call the V-space product form. The *V-space product form* consists of an upper triangular factor U_d and a symmetric positive definite matrix V such that

$$D = U_d^{\mathrm{T}} U_d \text{ (Cholesky factor)}$$

$$X = U_d^{\mathrm{T}} V U_d, \;\; V = U_d Z U_d^{\mathrm{T}}.$$

In fact, the idea to consider (U_d, V) as the iterates in a primal-dual method was proposed for other reasons already in 1995 by Sturm and Zhang [20]; this paper extended the V-space framework from linear to semidefinite programming, thereby obtaining a new interpretation for the Nesterov-Todd direction [15, 16]. We will now explain with an example why the V-space factors convey more information than the (X, Z) iterates in floating point representation.

Consider a simple 2×2 example, with quantities that are typical for an iteration approaching the final stage:

$$U_d = \begin{bmatrix} 9000 & 1000 \\ 0 & 0.002 \end{bmatrix}, \quad V = 10^{-5} \times \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix}.$$

The matrix V is well conditioned and positive definite. The upper-triangular matrix U_d satisfies the property of a *stable U-factor*, defined as follows:

Definition 1 *A $\nu \times \nu$ upper triangular matrix U is called a* stable U-factor *if for all rows $i = 1, 2, \ldots, \nu$ it holds that*

$$u_{ii} \geq \max\{u_{ij} \mid j = i+1, i+2, \ldots, \nu\}. \tag{9}$$

See Lemma 8.6 in Higham [8] for a motivation of the above definition.

We computed in MATLAB 5.3, using IEEE double precision floating point arithmetic (we obtained the identical results on a Sun Sparcstation 4 and a Cyrix 686 based PC), the following matrices:

$$X = U_d^{\mathrm{T}} V U_d, \; Z = U_d^{-1} V U_d^{-\mathrm{T}}, \; D = U_d^{\mathrm{T}} U_d.$$

In this particular example, the computed X is completely accurate. Furthermore, all entries of the computed Z must be highly accurate as well, because U_d is a stable U-factor and all entries of Z are of the same order of magnitude; see Theorem 8.7 in Higham [8]. The computed entries in D are the floating point representations of the exact D matrix, which is easily calculated from U_d. Nevertheless, we will see that important information has been lost.

In exact arithmetic, we should have $X = DZD$, but if we numerically evaluate $\|X - DZD\|_2$ we find an error of more than 0.1. The error is due to heavy numerical cancellation in evaluating the matrix product DZD. Evaluating $\|X - U_d^{\mathrm{T}}(U_d Z U_d^{\mathrm{T}})U_d\|$ yields and error of more than 0.06, which is hardly any better. The error must be explained from heavy cancellation in evaluating $U_d Z U_d^{\mathrm{T}}$; we claim that the entries in X, Z and D are accurate, relative to their magnitude. The argument below will further illustrate this.

Defining u_{ij}'s and z_{ij}'s by

$$U_d = \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}, \quad Z = \begin{bmatrix} z_{11} & z_{12} \\ z_{12} & z_{22} \end{bmatrix}, \quad V = \begin{bmatrix} v_{11} & v_{12} \\ v_{12} & v_{22} \end{bmatrix},$$

we have from $V = U_d Z U_d^{\mathrm{T}}$ that

$$v_{22} = z_{22}u_{22}^2, \quad v_{12} = u_{22}(u_{11}z_{12} + u_{12}z_{22}), \tag{10}$$

and

$$v_{11} = z_{11}\left(u_{11} + \frac{z_{12}u_{12}}{z_{11}}\right)^2 + \frac{\det(Z)u_{12}^2}{z_{11}}. \tag{11}$$

In our numerical example, we have

$$\begin{cases} z_{11}z_{22} = 0.69444351852222, \ \sqrt{z_{11}z_{22}} = .83333277777981 \\ z_{12}^2 = 0.69444351851883, z_{12} = -.83333277777778 \end{cases}$$

so that $z_{11}z_{22} - z_{12}^2 = 3.395 \times 10^{-12}$ suffers from numerical cancellation. However, we can compute $\det(Z)$ accurately from the U_d and V factors, viz.

$$\det(Z) = \frac{\det(V)}{\det(U_d)^2} = 3.395061728395063 \times 10^{-12}. \tag{12}$$

Let $\hat{V}$ denote the V-matrix that is computed in floating point arithmetic from (10)–(12). Then the error measure $\|X - U_d^{\mathrm{T}}\hat{V}U_d\|$ evaluates to less than 2×10^{-7}.

The above example illustrates some important points. First, the NT-scaling D cannot be accurately computed from (8) when Z gets ill-conditioned. In fact (8) fails to hold even for an accurate floating point representation of the actual D. The error $\|X - DZD\|$ is large because the contribution of the large eigenvalue in D, and therefore the majority of the significant digits in D, is almost canceled out when multiplying with Z. Second, the contributions of small eigenvalues in X, Z and D have low accuracy if X, Z and D are stored as matrices of floating point entries. Therefore, we should store and update X, Z and D in product form. Accurate results can be obtained from the *V-space product form*, provided that the iterates stay reasonably well centered so that V is well conditioned, and U_d is a stable U-factor. The latter can always be achieved by using pivoting [8].

4 Maintaining the V-Space Product Form

It is clear that the Nesterov-Todd scaling point D can easily be computed from the V-space factor U_d, viz. $D = U_d^{\mathrm{T}}U_d$. Now suppose that we can solve system (7) accurately. It is then straightforward

to compute the primal and dual directions ΔX and ΔZ, and the new iterate $(X^{\rm new}, y^{\rm new}, Z^{\rm new})$ as in (4). However, we will then face numerical problems in the next iterate, since the new NT-scaling $D^{\rm new}$ cannot be accurately computed from the floating point representations of the entries in ill-conditioned matrices $X^{\rm new}$ and $Z^{\rm new}$, as discussed before. Therefore, we propose to take the step in a scaled space, in which the primal and dual solutions remain (locally) well conditioned. The scaled search direction, denoted $(\overline{\Delta X}, \overline{\Delta Z})$, is implicitly defined as

$$U_d^{\rm T}\overline{\Delta X}U_d = \Delta X, \quad \overline{\Delta Z} = U_d \Delta Z U_d^{\rm T}.$$

In fact, since we use only the scaled search directions, it is not necessary to compute ΔX. Namely, once Δy and ΔZ have been computed we let $\overline{\Delta Z} = U_d \Delta Z U_d^{\rm T}$. Then, instead of solving Δx from

$$\Delta x + P(d)\Delta z = -x,$$

we solve $\overline{\Delta X}$ from

$$\overline{\Delta X} + \overline{\Delta Z} = -V.$$

Given the scaled search direction, we arrive at the new iterate under the current scaling,

$$\bar{X}^{\rm new} = V + t\overline{\Delta X}, \; \bar{Z}^{\rm new} = V + t\overline{\Delta Z}.$$

Since V is well conditioned, we can in principle control the step length t in such a way that $\bar{X}^{\rm new}$ and $\bar{Z}^{\rm new}$ are also well conditioned. However, in practice we will give priority to fast convergence, so that the conditioning of $\bar{X}^{\rm new}$ and $\bar{Z}^{\rm new}$ can be poor on those iterates where the rate of linear convergence is close to zero. The procedure for updating U_d will then be less accurate. Such inaccuracy can possibly lead to larger residual vectors that are included in a self-dual or infeasible interior point formulation; however, the increase will usually be offset by a reduction in the same residuals that is caused by the small (hence fast) rate of linear convergence.

The procedure to update the U_d and V factors is as follows:

Procedure 1 (Update of V-factors)

Input: $\bar{X}^{\rm new}$, $\bar{Z}^{\rm new}$, U_d.

Output: Upper triangular matrix $U_d^{\rm new}$ *and positive definite matrix* $V^{\rm new}$ *such that for*

$$X^{\rm new} := U_d^T \bar{X}^{\rm new} U_d, \quad Z^{\rm new} := U_d^{-1} \bar{Z}^{\rm new} U_d^{-T},$$

we have

$$X^{\rm new} = (U_d^{\rm new})^T V^{\rm new} U_d^{\rm new}, \quad V^{\rm new} = U_d^{\rm new} Z^{\rm new} (U_d^{\rm new})^T.$$

1. *Compute an upper triangular matrix* $\bar{U}_x^{\rm new}$ *as the Cholesky factorization of* $X^{\rm new}$, *i.e.*

$$\bar{X}^{\rm new} = (\bar{U}_x^{\rm new})^T \bar{U}_x^{\rm new}.$$

2. *Let* $W := \bar{U}_x^{\rm new} \bar{Z}^{\rm new} (\bar{U}_x^{\rm new})^T$; *compute an orthogonal matrix* Q_w *and a diagonal matrix* $\Lambda_v^{\rm new}$ *as the symmetric eigenvalue decomposition of* $W^{1/2}$, *i.e.*

$$W = Q_w (\Lambda_v^{\rm new})^2 Q_w^T.$$

3. *Let* $T := (\Lambda_v^{\text{new}})^{-1/2} Q_w^T$*; compute an orthogonal matrix* Q_v^{new} *and an upper triangular matrix* R *as the QR-factorization of* T*, i.e.*

$$T = (Q_v^{\text{new}})^T R.$$

4. *OUTPUT:*

$$U_d^{\text{new}} = R\bar{U}_x^{\text{new}} U_d,$$
$$V^{\text{new}} = Q_v^{\text{new}} \Lambda_v^{\text{new}} (Q_v^{\text{new}})^T.$$

The above procedure does not guarantee in itself that U_d^{new} is a stable upper triangular factor; hence a re-ordering of rows and columns may be necessary. A few Givens rotations will make the U_d^{new}-matrix upper-triangular in the new pivot ordering; see e.g. Dennis and Schnabel [4] for this technique.

Theorem 1 *Procedure 1 is correct.*

Proof. The proof is straightforward. We have

$$R^{\text{T}} V^{\text{new}} R = T^{\text{T}} \Lambda_v^{\text{new}} T = Q_w Q_w^{\text{T}} = I,$$

so that

$$(U_d^{\text{new}})^{\text{T}} V^{\text{new}} U_d^{\text{new}} = U_d^{\text{T}} (\bar{U}_x^{\text{new}})^{\text{T}} \bar{U}_x^{\text{new}} U_d = X^{\text{new}}.$$

Furthermore, we have

$$W = Q_w (\Lambda_v^{\text{new}})^2 Q_w^{\text{T}} = T^{-1} \Lambda_v^{\text{new}} T^{-\text{T}},$$

so that

$$U_d^{\text{new}} Z^{\text{new}} (U_d^{\text{new}})^{\text{T}} = R\bar{U}_x^{\text{new}} \bar{Z}^{\text{new}} (\bar{U}_x^{\text{new}})^{\text{T}} R^{\text{T}} = RWR^{\text{T}} = RT^{-1} \Lambda_v^{\text{new}} T^{-\text{T}} R^{\text{T}} = V^{\text{new}},$$

where we used that $RT^{-1} = R((Q_v^{\text{new}})^{-1} R)^{-1} = Q_v^{\text{new}}$. **Q.E.D.**

5 Building the Scaled Normal Equations System

In order to compute Δy from (7), we need to obtain the Cholesky factorization of $AP(d)A^{\text{T}}$. It is well known that this can be achieved by a QR-factorization of the matrix $P(d)^{1/2} A^{\text{T}}$, without building the matrix $AP(d)A^{\text{T}}$ itself. However, in most practical models we see that A is a very sparse matrix whereas $P(d)^{1/2} A^{\text{T}}$ is dense. Furthermore, the number of rows in A is typically *much* larger than the number of columns. Due to these properties, it is often computationally cheaper to compute $AP(d)A^{\text{T}}$ than $P(d)^{1/2} A^{\text{T}}$, both in terms of storage and number of operations. Therefore, the matrix $AP(d)A^{\text{T}}$ is explicitly formed in all implementations of the interior point method that the author is aware of.

The (i,j)th-entry in the matrix $AP(d)A^{\mathrm{T}}$ is

$$a_i^{\mathrm{T}} P(d) a_j = \text{ tr } A_i D A_j D. \tag{13}$$

Sparsity in the A matrix can be exploited throughout the computation of the entries in $AP(d)A^{\mathrm{T}}$. For instance, one may apply the following scheme:

1. Let $T := DA_j$.
2. Compute $(TD)_{p,q}$ for those (p,q) where the (p,q)th entry in A_i is nonzero for some $i \in \{1,2,\ldots,j\}$.
3. Compute $\text{ tr } A_i(TD)$ for $i = 1,2,\ldots,j$.

Unfortunately, some entries in the $AP(d)A^{\mathrm{T}}$ matrix that results from the above sparsity exploiting scheme may be inaccurate due to numerical cancellation. Cancellation happens on the ith diagonal when

$$\frac{a_i^{\mathrm{T}} P(d) a_i}{\sum_{j=1}^{n} |a_{ij}(P(d)a_i)_j|} << 1. \tag{14}$$

For example, on the instance `gpp100` in the SDPLIB set [3] of semidefinite programming instances, we find an iterate where

$$\begin{cases} \|A_1\|_F = 1.00 \cdot 10^2 \\ \|DA_1D\|_F = 3.65 \cdot 10^0 \\ \text{ tr } A_1DA_1D = \|U_d A_1 U_d^{\mathrm{T}}\|_F^2 = 1.76 \cdot 10^{-10} \end{cases}$$

In fact, in this example it holds that A_1 is the all-one matrix of size 100×100. Evaluating $\text{ tr } A_1(DA_1D)$ therefore amounts to adding 10,000 floating point entries, yielding a number whose magnitude is merely a 10^{-10} fraction of the original entries; hence there is massive cancellation. In this particular case where $A_1 = ee^{\mathrm{T}}$, e denoting the all-one vector, we have

$$\text{tr } A_1DA_1D = (e^{\mathrm{T}}De)^2 = \|U_d e\|_2^4,$$

providing a cheap formula to compute $\text{ tr } A_1DA_1D$ accurately. However, if A_1 is not given as a rank-1 matrix, we can accurately compute $\text{ tr } A_1DA_1D$ using the general formula

$$\text{tr } A_1DA_1D = \|U_d A_1 U_d^{\mathrm{T}}\|_F^2. \tag{15}$$

The disadvantage of the formula in (15) is that it will hardly benefit from possible sparsity in the A_1 matrix. Therefore, we should use (15) only if the ratio in (14) indicates massive cancellation.

6 Other Sources of Numerical Problems

Consider again the system (7), i.e.

$$AP(d)A^{\mathrm{T}}\,\Delta y = b.$$

In order to numerically determine Δy, we will first build the $AP(d)A^{\mathrm{T}}$ matrix, and then decompose it into its Cholesky factors. In fact, we will decompose the matrix as

$$AP(d)A^{\mathrm{T}} = L\Theta L^{\mathrm{T}},$$

where Θ is a positive definite diagonal matrix, and L is a lower triangular matrix for which $l_{ii} = 1$, $i = 1, 2, \ldots, m$.

Define $\Delta\tilde{y} := L^{\mathrm{T}}\Delta y$ and $\tilde{r} := L^{-1}b$. Pre-multiplying both sides in (7) with L^{-1}, we obtain the system

$$\Theta\Delta\tilde{y} = \tilde{r}, \tag{16}$$

where Θ is the diagonal matrix in the $L\Theta L^{\mathrm{T}}$ factorization. The solution to (16) is of course

$$\Delta\tilde{y}_i = \frac{\tilde{r}_i}{\theta_i} \text{ for } i = 1, 2, \ldots, m.$$

The computations involving the unit-diagonal lower triangular matrix L and the diagonal matrix Θ can usually be carried out with reasonable accuracy. However, we should be careful on how numerical errors in Θ are promoted into errors in $\Delta\tilde{y}$ and consequently Δy. We will illustrate this issue with an example.

On the semidefinite programming instance `hinf5`, which is in a problem library maintained at New York University [17] and in SDPLIB [3], we find in one of the final iterations of SeDuMi [18] that

$$\theta = \begin{bmatrix} 5.99 \cdot 10^{7} \\ \vdots \\ 1.11 \cdot 10^{-8} \end{bmatrix}, \tilde{r} = \begin{bmatrix} -1.10 \cdot 10^{0} \\ \vdots \\ -1.41 \cdot 10^{-5} \end{bmatrix}.$$

so

$$\Delta\tilde{y} = \begin{bmatrix} -1.84 \cdot 10^{-8} \\ \vdots \\ -1.27 \cdot 10^{3} \end{bmatrix}.$$

It turns out that the last entry in $\Delta\tilde{y}$ is much larger than the other entries. This means that all entries in the vector Δy become relatively large, and that the dual solution has not converged yet. In fact, this behavior indicates that the supremum cannot be attained in the dual. This situation is interesting from a numerical point of view. Namely, the Δy search direction is to a large extent determined by the contribution from the $\Delta\tilde{y}_m$ entry, which depends critically on $\theta_m \approx 1.11 \cdot 10^{-8}$. However, we can see from the θ_1 entry that the original entries in the $AP(d)A^{\mathrm{T}}$ matrix were of size 10^7. Hence, the θ_m entry suffers from heavy numerical cancellation and is in fact totally inaccurate. Therefore, the search direction as a whole becomes totally unreliable.

The phenomenon that some entries in θ suffer massive cancellation as $AP(d)A^{\mathrm{T}}$ becomes ill-conditioned, is already known from the interior point method for solving (degenerate) linear programs. In the setting of linear programming, the issue is resolved by setting the corresponding entry in $\Delta\tilde{y}$ to zero, i.e. setting $\Delta\tilde{y}_m = 0$ in the above example. The error that we make in solving (16) is then of size $|\tilde{r}_m|$. If the iterates converge, then the real Δy is small in the final stages, and so is $\Delta\tilde{y} = L^{\mathrm{T}}\Delta y$. The fact that θ_m is small then implies that $\tilde{r}_m$ is tiny, and hence the approach

problem	b*y	relerr	problem	b*y	relerr
hinf1	-2.032606992E+000	4E-011	hinf14	-1.299015801E+001	2E-008
hinf2	-1.096707467E+001	7E-010	hinf15	-2.483769673E+001	1E-005
hinf3	-5.694110666E+001	4E-009	ladder1	-2.353439750E+001	1E-014
hinf4	-2.747639132E+002	3E-010	ladder2	-2.346762362E+001	3E-012
hinf5	-3.622325357E+002	2E-007	truss1	8.999996315E+000	6E-012
hinf6	-4.489306798E+002	5E-009	truss2	1.233803564E+002	3E-013
hinf7	-3.908146948E+002	2E-006	truss3	9.109996208E+000	4E-011
hinf8	-1.161468835E+002	2E-008	truss4	9.009996291E+000	2E-011
hinf9	-2.362492581E+002	2E-010	truss5	1.326356780E+002	7E-012
hinf10	-1.088046491E+002	2E-007	truss6	9.010014047E+002	5E-011
hinf11	-6.591291881E+001	1E-007	truss7	9.000014036E+002	8E-011
hinf12	-2.040510749E-002	1E-010	truss8	1.331145892E+002	1E-012
hinf13	-4.436602472E+001	6E-008			

Table 1: Results using SeDuMi 1.04Beta on NYU test set

of setting $\Delta\tilde{y}_m = 0$ is accurate. However, in the `hinf5` instance we see that $\tilde{r}_m$ is not tiny, and hence we cannot simply discard it.

A possible solution in this situation is to precondition the A-matrix to say $\tilde{A} := MA$, $\tilde{b} = Mb$, where M is an invertible matrix, such that $\tilde{a}_m^{\mathrm{T}} P(d) \tilde{a}_m$ is sufficiently small, thus avoiding cancellation during the Cholesky process. The preconditioner can be based on the L-factor from the previous iteration in the interior point method: simply set $M = I + e_m(e_m^{\mathrm{T}} L^{-1} - e_m^{\mathrm{T}})$, where e_m denotes the mth column of the identity matrix. The quantity $\tilde{a}_m^{\mathrm{T}} P(d) \tilde{a}_m$ can be computed with sufficient accuracy using the technique described in Section 5.

7 Experimental Results

The techniques as discussed in this paper have been implemented in SeDuMi [18], version 1.04. We evaluated our approach on the set of semidefinite programs collected at New York University [17]. As a measure of performance, we used the following quantity:

$$\texttt{relerr} = \max\{\frac{c^{\mathrm{T}}x - b^{\mathrm{T}}y}{1 + |b^{\mathrm{T}}y|}, \frac{[\lambda_{\min}(c - A^{\mathrm{T}}y)]_-}{1 + \|c\|_\infty}, \frac{\|Ax - b\|_\infty}{1 + \|b\|_\infty}\},$$

where $[\alpha]_- := \max(-\alpha, 0)$ denotes the negative part. The results are listed in Table 1.

We can see from Table 1 that there are still a few instances for which the attained accuracy is poor. This can possibly be explained from the ordering strategy in the Cholesky decomposition of $AP(d)A^{\mathrm{T}}$. SeDuMi uses a minimum degree ordering, and the same ordering is used throughout the process. For the instances in [17], the matrix $AP(d)A^{|T}$ is in fact fully dense. The Cholesky factors that are obtained using this ordering may not be stable. In fact, since the diagonal of L is all-one, the factor L is stable if and only if $\max_{i,j} |l_{ij}| \leq 1$. However, in the last step before termination in SeDuMi, we obtain the following results:

problem	relerr	$\max_{i,j} \lvert l_{ij} \rvert$
hinf12	1E-010	6E+001
hinf13	6E-008	6E+009
hinf14	2E-008	1E+009
hinf15	1E-005	6E+006

We see that the use of a fixed pivot ordering may result in highly unstable L-factors, thus leading to substantial numerical errors in the Δy computation. Therefore, it may be necessary to reorder the pivots. Such a reordering is certainly not attractive if the $AP(d)A^{\mathrm{T}}$ matrix is sparse. We leave this issue for future research.

Some of the problem instances that we discussed are also in the DIMACS set of semidefinite and second order cone programs, viz. `hinf12`, `hinf13`, `truss5`, and `truss8`. On all `hinf` problems except 7,9 and 15, SeDuMi found a strictly feasible dual solution y, i.e. $C - \sum_{i=1}^{m} y_i A_i$ is positive definite. This implies that $b^{\mathrm{T}}y$ is an exact lower bound on the optimal value of (2). Surprisingly, the computed primal solution x has a better objective value, i.e. $c^{\mathrm{T}}x < b^{\mathrm{T}}y$, for all `hinf`-problems. The computed x solutions are positive semidefinite, so the negative duality gap can only be explained from the constraint violation $\|Ax - b\|$. The following table displays the exact lower bound $b^{\mathrm{T}}y$ as computed by SeDuMi, together with the final primal objective values as computed by SeDuMi [18], SDPT3 [25] and SDPA [6]. The results on SDPT3 and SDPA are cited from Mittelmann [14].

	hinf12	hinf13
$b^{\mathrm{T}}y$	-0.020405	-44.366
SeDuMi $c^{\mathrm{T}}x$	-0.023141	-44.368
SDPT3 $c^{\mathrm{T}}x$	-0.786	-46.64
SDPA $c^{\mathrm{T}}x$	-0.298	-47.28

For `hinf12`, the x-solution computed by SeDuMi has a constraint violation of $\|Ax-b\|_2 =$`4E-10`. One may be surprised that a substantial negative duality gap can be obtained with such a small constraint violation. In fact, it was shown in [24, 19] that given a solution x with a constraint violation $\|Ax - b\|$, and a quantity $\delta > 0$, there cannot exist a dual solution y with an objective value $b^{\mathrm{T}}y \geq c^{\mathrm{T}}x + \delta$, unless $\|y\|_2 \geq \delta / \|Ax - b\|_2$. This implies for the y solution computed by SeDuMi that

$$\|y\|_2 \geq \frac{b^{\mathrm{T}}y - c^{\mathrm{T}}x}{\|Ax - b\|} \approx \frac{0.02736}{\texttt{4E-10}} > \texttt{6E7}.$$

Indeed, the actual size of the computed y solution is $\|y\|_2 =$ `1E9`.

8 Conclusions

We have identified various sources of numerical problems in the interior point method, that are specific to semidefinite programming. Since these specific issues have not been dealt with before, we faced a heavy numerical challenge in our effort to develop a reliable implementation of the interior point method for semidefinite programming.

An important technique to avoid numerical cancellation in computing the Nesterov-Todd scaling has been proposed in this paper: the V-space product form. The V-space product form was already

used as an elegant approach to derive the Nesterov-Todd direction in Sturm and Zhang [20], but the importance of this product form for numerical computing has not been recognized before. Moreover, we have proposed an effective algorithm to update the V-space factors in an interior point framework.

We have seen that heavy numerical cancellation is possible while building the $AP(d)A^{\mathrm{T}}$ matrix, and how such cancellation can be detected and avoided. It was shown that unlike in linear programming, the ill-conditioning of the $AP(d)A^{\mathrm{T}}$ matrix can be harmful, and a possible remedy has been proposed. The amount of extra work and storage depends on the number of constraints in which harmful cancellation occurs during the Cholesky factorization. Finally, we have shown evidence that pivoting may sometimes be necessary in the Cholesky factorization of the $AP(d)A^{\mathrm{T}}$ matrix. However, more research needs to be done on this issue.

References

[1] F. Alizadeh, J.A. Haeberly, and M. Overton. Primal–dual interior–point methods for semidefinite programming: convergence rates, stability and numerical results. *SIAM Journal on Optimization*, 8(3):746–768, 1998.

[2] T. Ando. Concavity of certain maps and positive definite matrices and applications to Hadamard products. *Linear Algebra and its Applications*, 26:203–241, 1979.

[3] B. Borchers. SDPLIB 1.2: a library of semidefinite programming test problems. Technical report, New Mexico Tech Mathematics Faculty, USA, 1999.

[4] J.E. Dennis and R.B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations.* Prentice-Hall, 1983.

[5] J. Faraut and A. Korányi. *Analysis on Symmetric Cones.* Oxford Mathematical Monographs. Oxford University Press, New York, 1994.

[6] K. Fujisawa, M. Fukuda, M. Kojima, and K. Nakata. Numerical evaluation of SDPA (semidefinite programming algorithm). In J.B.G. Frenk, C. Roos, T. Terlaky, and S. Zhang, editors, *High Performance Optimization*, pages 267–301. Kluwer Academic Publishers, 2000.

[7] D. Goldfarb and K. Scheinberg. Interior point trajectories in semidefinite programming. *SIAM Journal on Optimization*, 8:871–886, 1998.

[8] N.J. Higham. *Accuracy and stability of numerical algorithms.* SIAM, Philadelphia, 1996.

[9] R.A. Horn and C.R. Johnson. *Matrix analysis.* Cambridge University Press, Cambridge, New York, 1985. Corrected reprint 1990.

[10] S. Kruk. *High accuracy algorithms for the solutions of semidefinite linear programs.* PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, February 2001.

[11] Z.-Q. Luo, J.F. Sturm, and S. Zhang. Superlinear convergence of a symmetric primal–dual path following algorithm for semidefinite programming. *SIAM Journal on Optimization*, 8(1):59–81, 1998.

[12] I.J. Lustig, R.E. Marsten, and D.F. Shanno. Computational experience with a primal–dual interior point method for linear programming. *Linear Algebra and its Applications*, 152:191–222, 1991.

[13] S. Mehrotra. On the implementation of a primal–dual interior point method. *SIAM Journal on Optimization*, 2:575–601, 1992.

[14] H.D. Mittelmann. Independent benchmark results. `http://plato.la.asu.edu/dimacs.html`, 2001.

[15] Y. Nesterov and M.J. Todd. Self–scaled barriers and interior–point methods for convex programming. *Mathematics of Operations Research*, 22(1):1–42, 1997.

[16] Y. Nesterov and M.J. Todd. Primal–dual interior–point methods for self–scaled cones. *SIAM Journal on Optimization*, 8:324–364, 1998.

[17] M.L. Overton. Test problems: LMI, truss and Steiner tree problems. Located at URL `http://cs.nyu.edu/cs/faculty/overton/sdppack/sdppack.html`, 1997.

[18] J.F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11–12:625–653, 1999. Special issue on Interior Point Methods (CD supplement with software).

[19] J.F. Sturm. Theory and algorithms for semidefinite programming. In J.B.G. Frenk, C. Roos, T. Terlaky, and S. Zhang, editors, *High Performance Optimization*, pages 3–196. Kluwer Academic Publishers, 2000.

[20] J.F. Sturm and S. Zhang. Symmetric primal–dual path following algorithms for semidefinite programming. *Applied Numerical Mathematics*, 29:301–315, 1999.

[21] M.J. Todd. A study of search directions in interior–point methods for semidefinite programming. *Optimization Methods and Software*, 11–12:1–46, 1999.

[22] M.J. Todd. Semidefinite optimization. Technical report, Cornell University, Ithaca, New York, USA, December 2000.

[23] M.J. Todd, K.C. Toh, and R.H. Tütüncü. On the Nesterov–Todd direction in semidefinite programming. *SIAM Journal on Optimization*, 8(3):769–796, 1998.

[24] M.J. Todd and Y. Ye. Approximate Farkas lemmas and stopping rules for iterative infeasible–point algorithms for linear programming. *Mathematical Programming*, 81:1–21, 1998.

[25] K.C. Toh, M.J. Todd, and R.H. Tütüncü. SDPT3 - a MATLAB package for semidefinite programming. version 1.3. *Optimization Methods and Software*, 11–12:545–581, 1999. Special issue on Interior Point Methods (CD supplement with software).

[26] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38:49–95, 1996.

[27] Y. Ye, M.J. Todd, and S. Mizuno. An $O(\sqrt{n}L)$-iteration homogeneous and self–dual linear programming algorithm. *Mathematics of Operations Research*, 19:53–67, 1994.